Setup, Synthesis and Testing the Opencores I2C Master Only Controller on a Leon SPARC with Xilinx Virtex Board

A Comprehensive, Step-By-Step Guide

Filip Miletic
Student Trainee
DIMES DDTC

filmil@galeb.etf.bg.ac.yu

This is where we describe making a prototype I2C device for the Leon SPARC.

Introduction

* *This document may be incomplete. If you are in position to make ammendments to it (e.g. you are continuing this work), please do so. Feel free to forward all issues to me.*

This document is aimed at being a step by step guide for a newcomer to be able to synthesize, compile and run all the software and hardware needed to have an operating prototype I2C device, with an archive we provide. That is, if you have this document standalone, you are not going to have any direct results.

The tutorial assumes the Windows environment, as it is most likely to cause problems. However most advices should apply for other platforms as well.

Prerequisites

Apart from the XSV Board, there is a number of items (hardware, programs and utilities) you need to have installed in order to proceed. They are numbered here.

- *The LK404-AT Liquid Crystal Display*

  This is the critical component in the setup. Critical meaning that the example is matched with the display.

Other I2C devices can be used as well, but you should then investigate ways to write and read from the device. The source code provides ways for you to choose the device address with which to communicate and also provides routine for reading and writing. However, in that case you need to invest some time to understand the source, which should not take too long to do.

- *Synthesis tools*

  You need to have a tool to translate the design language (*VHDL* in this case) to *EDIF* format. We used Synplify Pro 7 from Synplicity, but I suppose you could decide to go for another synthesis tool.

- *Xilinx Foundation Tools*

  These tools you need in order to make a bit file compatible with Xilinx FPGA devices from the EDIF description produced by the synthesis tool.

- *Cygwin*

  This package is essential for the software tools to work properly as the whole Leon SPARC software bundle will run only under Cygwin.

- *GNU C compiler and binutils*

  These utilities can be downloaded from Gaisler Research website (http://www.gaisler.com)

- *XSV Tools*

  This is a set of tools that enable you to download the synthesized bitstreams and source files into the *XSV Board*. The programs within this set include both command line and menu driven utilities.

- *Serial Communication Program*

  You need a serial communication program to be able to receive prompts from LEON's UART. We used Tera Term. For some reason, the Windows default program, Hyper Terminal did not want to work.

Board Setup

First prerequisite for succesful experiments is having the *FPGA* hardware working properly. For this, it is more than useful to equip yourself with the *XSV Board User Manual* which is available from Xess Corporation (http://www.xess.com).

> **Setting the clock frequency:** You need to set up the clock frequency before you begin. Leon SPARC is a complex system and it is not possible for it to work if timing constraints are not met. Please refer to the Manual for details on how to do this.

What follows is the checklist of needed items.

- It is a good idea to decouple the I2C bus you are experimenting with and the XSV Board with a line buffer. For this purpose we used a small board containing the I2C buffer chip from 82715 from Philips Semiconductor (http://www.philipssemiconductor.com).
- The XSV Board uses a programmable oscillator with a default frequency of 100MHz. You must reprogram the oscillator and set it to at most 25MHz.
- Two or three cables are needed, depending on your setup. First one is the 9VDC supply voltage cable. The second one is the parallel cable to connect the XSV Board to the computer which will be programming it. Third is, in fact, used to connect to the I2C buffer board, something you are likely to have to do if you have a dedicated I2C cable or, for that matter, an PS/2 cable you misuse for I2C. See section Cabling for details on how to do this.
- When you want to change the default clock frequency, you must do a series of steps involving shorting jumpers 36 and 21. You must not forget to return this jumper to the normal runtime position once the clock frequency setting is done. The clock setting utility, `gxssetclk` will warn you about the correct sequence to accomplish this.

Cabling

You will have to connect several cables to the XSV Board and your I2C device for a successful setup.

The cables are, in order:

1. An RS232 9pin female/female cable to connect the UART to the host computer.
2. The parallel cable connecting the XSV Board to the host computer;
3. A custom made cable, in our case a pair of wires which attaches to the I2C buffer.

Connecting the buffer can be tricky. You need to have wires attached to one of the pins of the piggyback connector on the XS Board. That is the one with lots of pins, as found on hard drive controllers, for instance.

The pins alotted to the I2C interface are as follows:

- The SDA connects to pin number 1 on the XSV Board PS/2 connector. This information is based upon the pinout I found at this place (http://www.baber.com/baber/products/ps2_mouse_port_5_pin1.htm). There are different pinouts available, with different pin numberings.
- The SCL connects to pin number 5 on the XSV Board PS/2 connector.
- The VCC of the buffer must be connected to pin +5V on the XSV Board piggyback pin set.
- The GND of the buffer must be connected to pin GND on the XSV Board piggyback pin set.

Rolling your own I2C buffer: In our case, the buffer circuit was fitted with the pull-up resistors. Without these resistors, the I2C bus will not work.

Be sure not to forget a pair of pull-ups on either side of the buffer, in case you are making your own buffer. Therefore, you need 4 resistors in total, one for each of the I2C lines for each of the two sides of the buffer. We used 4 time 2k2 resistors. They are distinguished by having three *orange* stripes on the body.

The integrated circuit used for the buffer has two functionally equivalent sets of inputs, one set for the *local* and the *buffered* bus. Although the buffer will forward signals from either side to the other, the *buffered* output side in intended to be attached to the cables and has better driving capacity. You should use this side to attach to remote devices.

Installation

Let us assume that what you have in the beginning is a sort of an archive containing the files needed for synthesis.

You should unpack the archive in a directory of your choice. The archive should unpack in a directory of its own, with a number of subdirectories within it. In case you get a flat decompress, with all the files in the same directory, something has gone wrong. You might want to continue working that way, but it really is not a good idea.

Let us call the root directory of your installation `$ROOTDIR`. References to this name will be made throughout this document.

Synthesis

In order to synthesize the *VHDL* model, you need to process the *VHDL* design with a number of programs. The typical necessary steps are mentioned in this section.

Running *VHDL* Synthesis Tool

You need to open the synthesis tool and import all the required files. Better still, you might even be able to do it from within the command line.

The default project file is supplied for *Synplify*. You can find it in the directory `$ROOTDIR/syn`. You should be able to open it from within the *Synplify* development environment, and be free of errors.

It should now be possible to click on the Run button and start translation.

Changing targets: The project file is preset for the XSV Board model. In case you have another board to work with, you will have to review the synthesis parameters.

The translation should produce a file named `$ROOTDIR/syn/xess-leon.edf`. This is an *EDIF* file required as input to the *Xilinx Design Manager*.

Creating Bitstream File

The *EDIF* file that we created by using Synplify and the instructions from the previous section needs to be converted into a bitstream file (extension `.BIT`) in order to be downloaded onto the *FPGA* device.

The tool we used for the conversion is the Xilinx Design Manager from the Xilinx Foundation Tools.

The first step is invoking the Xilinx Design Manager either from the Start menu or from the command line.

> **Non-windows users note:** You may find the startup method to be different for your system.

You should now open the Xilinx Design Manager project file. This file is distinguished by the extension `.XPJ`. A default project file is provided for Design Manager. This file contains all the necessary pieces of information fro successful bitstream generation. We are therefore describing the steps needed to use this file to obtain the *FPGA* bitstream.

Select the File⟶Open (**Ctrl-O**) option. A dialog box will appear offering possibility to either choose one of the available projects from the list or select an existing project that is not in the list.

From the Open dialog box, select the Browse option. Find the file by the name `$ROOTDIR/syn/xproj/xess-leon.prj` and open it.

To complete bitstream generation you need to select Design⟶Implement menu option and wait for the design manager to finish.

If you decide to create a new project.

Do not forget to include the right constraints file. It is used to make physical mapping from *VHDL* signal names to actual *FPGA* pins. You absoulutely *need* to have an appropriate constraints file.

A constraints file suitable for use in this example is provided as `$ROOTDIR/syn/xess-BonPS2.ucf`.

If you decide to use the Design Manager project file provided with the project, you need not worry about the constraints file as it is included automatically.

The bitstream generation process should yield a file named `$ROOTDIR/syn/xess-leon.bit`. This file contains encoded description of the *FPGA* logic configuration corresponding to the design we began with, and readable by the programmer utility.

Making the Software

If you completed the steps from the previous section with success, you should be ready to make what is remaining: the software.

Provided that you have GNU C and GNU Make installed, the make procedure should be no more than changing to the directory `$ROOTDIR/src` and typing **make** at the command line.

The file resulting from this compilation is `$ROOTDIR/src/i2c_test_leon.srec`. This file contains the SREC formatted program for Leon.

Now, what in fact is in the example? The example uses the I2C to form a looopback terminal to an 40x4 character LCD device under the model name LK404-AT, made by Matrix Orbital (http://www.matrixorbital.com). This display has an I2C interface to communicate with the host device, and can also have a regular PS/2 keyboard attached so as to form a full featured terminal.

> **Data format for the LCD:** Firstly, the I2C device has an I2C bus address allocated. The 7-bit address is 0x28. This address should be supplemented with the eighth bit from the LSB side, depending upon the intended data transfer direction. Please refer to the appropriate manual for detailed description.
>
> The data sent to the display is a stream of 8-bit values. In case you just want to display pure text, sending the ASCII codes of the characters will be enough. There are some reserved values which enable you to control the appearance of the text.
>
> The display can send the characters typed on its keyboard via the I2C interface. The only way to extract these characters from the display is via a master read from the controlling host (e.g. the microcontroller). The data format is as follows. In case no characters are available in the display's buffer, the display will return a NUL ascii value (0x00). In case characters are present in the buffer, the low 7 bits are returned in the 7 lower bits of the data read from the device. The bit 8 is set in case there are more characters pending, and is reset otherwise.
>
> The LCD buffer is circular and is about 10 characters long. In case of buffer overrun, the old buffer contents are lost. Therefore the software must ensure timely emptying of the buffer.

Downloading Design

You should have the files named `$ROOTDIR/syn/xess-leon.bit` and `$ROOTDIR/syn/dwnldpar.svf` handy for this.

You need to have either `xsload` or `gxsload` utility. We will describe only the former, which is the command line interface to the downloading proper.

If you have not downloaded any bitstreams to the XSV board before, you need to configure its CPLD chip so that the board is correctly set up for programming. This needs to be done once after the clock frequency has been changed. The CPLD settings are remembered across runs.

To do this, issue the command:

xsload $ROOTDIR/syn/dwnldpar.svf

This download takes a while.

In case you have the utilities configured correctly, you should now be able to issue a command

xsload $ROOTDIR/syn/xess-leon.bit

which will complete the download for you. The download might take some time to complete.

You should be able to see the progress report on the monitor. If no errors occur, the board should be programmed correctly.

Startup

All the interaction with the developer is achieved via the serial link.

The default link parameters are: 38400bps, 8 data bits, no parity, 1 stop bit. Note that the correct bit rate is obtained in compile time from the supplied clock frequency. Thus, if you changed the clock frequency, you should note that in the Leon source. The appropriate place to look for these settings is the file `$ROOTDIR/leon/target.vhd`, a section on UART.

Leon has its own reset generator and starts working immediately upon programming. The boot message, of the form of:

```
LEON-1 etc...
```

should appear on the terminal screen. You might want to press the **RESET** button on the XS Board to provoke re-initialization.

If everything is installed properly, you should see a prompt printed on the terminal and have all the keyboard input of the terminal looped back to the screen. The prompt waits for you to send a Motorola SREC file. You can accomplish this by selecting the option to send an ASCII file.

You can build the example programs by changing to directory `$ROOTDIR/leon-opencores-i2c/src` and typing `make`. This should produce several SREC files.

(Non) Troubleshooting

I encountered no problems with the XSV Board. It is a reliable and sound design, as far as I was able to see.

File Layout Reference

What follows is a guide that will hopefully explain the files and directories comprising this design. We will first give an introductory note on the directories and their intended purpose.

Overview of Directories

Please note that I cannot describe all the detail of the Leon files. A good manual on the Leon is available for download from Gaisler Research (http://www.gaisler.com).

The design files that are not present in the original design of the Leon and were added by us are all supplied in the directory called `$ROOTDIR/leon-opencores/i2c`.

The following directories are present within the `$ROOTDIR/leon-opencores/i2c` directory.

- `doc`

  Contains the documentation you are now reading, in various formats. A PostScript documentation is available in the very directory. Other formats are present as subdirectories to this one.

The source document from which the documentation is generated is written in SGML and all modifications should go there.

- `src`

  Contains the C source code to the examples.

- `syn`

  This is the synthesis subdirectory. The files necessary for synthesis are put into this directory. This directory is an addition to Leon's original `$ROOTDIR/syn`.

- `tb`

* *This directory is empty for Opencores I2C on Leon.*

  Contains files necessary for building a testbench. It hosts additional testbench VHDL files and Tcl scripts which come handy if you need to simulate the design in ModelSim.

- `vhdl`

  Contains VHDL source files. Autonomous packages, i.e. those that can be used independently of the whole design are shown as subdirectories.

Overview of Files per Directory

- Directory `doc/`.

  `i2c_rev4.pdf`. This is the documentation for the *Openores* I2C master-only core used in this design. Please refer to it if you need a manual on internals of the *I2C* core.

  `leon-opencores.ps`. This is the PostScript version of the document you are reading.

  `html/`. This subdirectory contains the HTML version of the document, suitable for online reading. Please find file `index.html` which is the root document of that directory.

  `sgml/`. This subdirectory contains the SGML source file of this document. Please find the file `leon-opencores.sgml` if you wish to review the source text or make modifications.

- Directory `pre/`.

  `Makefile`. This is a tiny GNU Make script that will download the pre-built design to your board. Just type `make`.

  > **Requirements:** Please be aware that in order for the pre-built design to work, you need to have exactly the same setup as described in section on prerequisites.

`i2c_test_leon.srec`. This file is an Motorola SREC format of the program. It gets downloaded into the onboard RAM. This is the main example file for the loopback terminal.

`xess-leon.bit` contains a binary stream which is used to configure the FPGA.

- `dwnldpar.svf` contains a binary stream which is used to configure the CPLD.
- Directory `src`

  `Makefile` is a GNU Make script to produce the `i2c_test_leon.hex` file, the default example.

  `i2c_test_leon.c` is a top level source file for the example.

  > **Clock Frequency Setting:** The examples depend upon correct clock frequency setting in the source code. Please find the `#define CPU_FREQ` constant in the source code and verify it corresponds to the actual board frequency.

  `oc_i2c_master.h` is a header file containing the `#define`'s that are used in the examples above. It is worth considering when writing a library.

- Directory `syn`

  `xess-BonPS2.ucf`. This is a constraints file that helps Design Manager allocate physical pins for VHDL signals.

- Directory `tb`.

* *This directory is empty.*

- Directory `vhdl`.

  `xess.vhd`. This is a top level entity for building Leon on Virtex a board. It is already included in the main Leon source.

  `leon-2.4.0.diff`. This is a diff file that summarizes all changes made to the original Leon source to produce the modified version that is available here. This diff was already applied to the Leon source in `$ROOTDIR`.

  There exist two subdirectories here. What follows is the description of their contents.

  Directory `uclib` is the source of the uclib package for describing microcontroller chipsets in VHDL.

  - `uclib.vhd` is a package declaration providing components within the uclib package.
  - `address_decoder.vhd` is an implementation of the 8051 compatible address decoder.
  - `address_latch.vhd` is an implementation of the 8051 compatible address latch used to demultiplex address and data bus lines.

- `group4_decoder.vhd` is an implementation of the 8051 compatible address decoder of a group of successive addresses.
- `inout_register.vhd` is an implementation of the 8051 compatible bidirectional data register.
- `inout_register_val.vhd` is another implementation of the 8051 compatible bidirectional data register. This version exports the value as well.
- `out_register_val.vhd` is an implementation of the 8051 compatible output data register. This version exports the value as well.
- `led_inout_register.vhd` is an implementation of the 8051 compatible register for the LED.
- `memory_access.vhd` is an implementation of the 8051 compatible memory access arbitration unit.

Directory `opencores-i2c`.

- `i2c_master_*.vhd` are a set of files provided by *Opencores*.
- `i2c_device_top.vhd` is a wrapper for the original I2C device to adapt it to 8051 bus. This is a top level entity for the Opencores I2C device as used in this example.
- `tristate8.vhd` is a set of tristate drivers needed to provide access to the 8051 data bus.

Further Info

Should you need further asisstance or you have any questions as regards the documents that are exposed here, or you have any suggestions or corrections to supply, please feel free to write to the following address: <`filmil@galeb.etf.bg.ac.yu`>.